Some Research Directions in the Ptolemy Project[†]

H. John Reekie and Edward A. Lee
University of California at Berkeley
johnr@eecs.berkeley.edu, eal@eecs.berkeley.edu

The Ptolemy project studies modeling, simulation, and design of concurrent, real-time, embedded systems. The focus is on assembly of concurrent components. The key underlying principle in the project is the use of well-defined *models of computation* that govern the interaction between components. The software system Ptolemy II has been in public release for about nine months now, and includes a number of implementations of different models of computation. In this presentation, we outline some new research directions in this project, aimed at enhancing the level of productivity and verifiability of real-time and embedded systems.

[†] Presented at the Workshop on New Visions for Software Design and Productivity: Research and Applications, Vanderbilt University, Nashville Tennesee, December 12 – 14, 2001.

In this white paper, we briefly summarize the current state of the Ptolemy II software system [1], and outline some current research directions that we feel may be of interest to this community. One of the goals of Ptolemy II is to provide a theoretical and practical framework for defining and producing embedded software. Executable models are constructed under a *model of computation*, which can be thought of as the "laws of physics" that govern the interaction of components in the model. The choice of model of computation depends strongly on the type of model being constructed. Models of computation that have been implemented in the Ptolemy group include synchronous dataflow, finite state machine, continuous time, and synchronous-reactive models of computation. Other research groups at UCB have used Ptolemy II as a platform for research into other models of computation [4, 5].

One principle of the Ptolemy project is that the choice of models of computation strongly affects the quality of a system design. For example, in embedded systems, useful models of computation typically include the notions of concurrency and time. Embedded systems by their nature contain concurrent components and respond to external events. Choosing an inappropriate model of computation may lead the designer into a more costly or less reliable implementation.

A second key principle is the use of multiple models of computation constructed in a hierarchy of models. We believe that no single general-purpose model of computation is likely to deliver what designers need to model a complex embedded system. Modeling the diverse implementation technologies and their interactions is not reasonable within a homogeneous environment. Ptolemy II therefore supports the construction and interoperability of executable models that are built under a wide variety of models of computation.

Following sections describe some new areas of research that we are investigating within the Ptolemy II framework.

Interface theories

We are just beginning to explore Alfaro and Henzinger's *interface theories* [6] in the context of Ptolemy. Although Ptolemy is an excellent research and execution platform for embedded systems components, it does not directly support more abstract system specification. In particular, it does not support refinement of interfaces into sub-interfaces and components.

An interface theory consists of an interface algebra **A**, a component algebra **B**, and an implementation ◄ of **A** by **B**. Thus, **B** implements **A**, or **B** ◄ **A**. The interface algebra supports the key operations of composition, connection, and refinement, while the component algebra supports compositional implementation. Refinement of interfaces is contra-variant on inputs and outputs (for example, in the I/O interface algebra, a refinement of the interface can have fewer input ports and more output ports). Refinement of components, in contrast, is co-variant.

Examples of interface algebras include I/O interfaces, assume/guarantee interfaces, and port dependency interfaces. Although Ptolemy does have an implicit I/O interface algebra (its type-inference engine), it does not explicitly support the notion of different interface theories.

We are exploring the possibility of providing direct support for interface theories in Ptolemy. In one scenario, interface theories are akin to a model of computation in Ptolemy. That is, it may be possible for an implementer to provide a new interface theory, if the application domain and/or theoretical model require it. The process of system design would then include successive

refinement of the model within the interface algebra, and eventual execution of components in the appropriate component algebra – that is, within a suitable model of computation.

Modal and higher-order models

Ptolemy II is currently a first-order language. In addition, most models produced in Ptolemy have static structure – that is, the model does not change structure during execution. Ptolemy does have the ability to mutate models during execution, but this facility is somewhat *ad-hoc* and outside the bounds of the formal theories on which much of Ptolemy is based [7]. We are interested in pushing the boundaries of expressivity in Ptolemy with modal and higher-order models, together with appropriate formalisms that describe this type of operation.

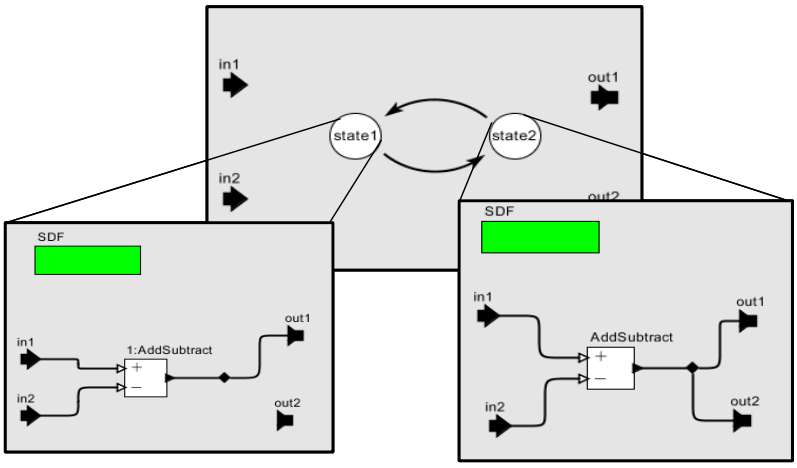

Figure 1. A modal model (from Eker and Lee). Each state in the upper model represents a different connection of components in the lower model

In a modal model, a model such as a dataflow or continuous time model is embedded within a model of a "control" nature, such as a state machine or synchronous-reactive model. Ptolemy currently supports embedding of models of diverse MoCs, but each state of the state machine (for example) represents a different model internally. In a modal model, each state of the machine represents a different set of connections of a single dataflow (for example) model. Thus, the construction is not strictly hierarchical, as state persists in the lower level model as the upper level model proceeds through its state space (Figure 1).

In Ptolemy Classic (the first version of Ptolemy), a *higher-order function* was a component that took the name of another component as a parameter, and replicated that component together with an appropriate communication network. This was based on earlier work that drew parallels between dataflow and functional programming systems [3]. Figure 2 illustrates a Ptolemy II model that would benefit from a feature like this, and which could be expressed with two higher-order functions. This model is being developed by Steve Neuendorffer for Joszef Ludwig (of Lawrence Berkeley Labs), for analyzing signals from neutrino detection arrays suspended in the Antarctic sea. A more realistic model would have 64 x 64 arrays of detectors.

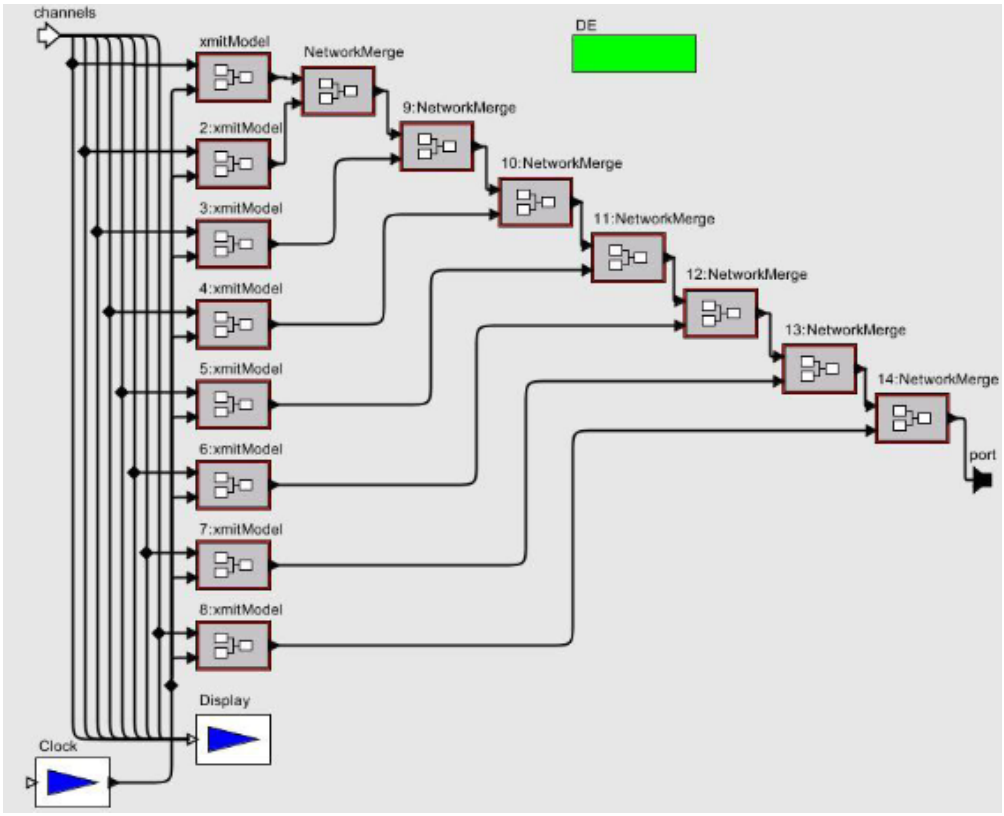

Figure 2. A model that needs higher-order functions. This system could be represented as two higher-order actors, parameterized by xmitModel and NetworkMerge actors

We are exploring re-introducing a facility like this in Ptolemy II, together with compilation technology to make execution of large expansions feasible. We would, however, like it to be part of a more general theoretical framework, in which the "higher-order-ness" does not necessarily require static knowledge of parameters such as (in this example) the number of detectors in an array. This theoretical framework will take account of the varying behavior of such higher-order constructs in the presence of interface theories and different models of computation.

A more general form of higher-order behavior is exemplified by work done by Esser and Janneck for design space exploration [2]. (Although this work was done in the context of the Moses system, Janneck is currently a post-doctoral researcher in the Ptolemy group.) In this context, a model can be a first-class object. To perform design-space exploration, the designer constructs two models: one to represent the search strategy, and one to represent the model being optimized. The latter system is passed around the former system as a token, and parameters are fed into and retrieved from this "model under test" to assist the "explorer" model in its task (Figure 3).

Conclusion

We have outlined some research directions under way in the Ptolemy project. As always, we strive for a pragmatic blend of formalism and implementation. If successful, these research fundamentals could become a simple architecture that serves as the basis for a distributed experimental platform for further research into component-based design for embedded systems. This would be done in the context of the NEPHEST project.

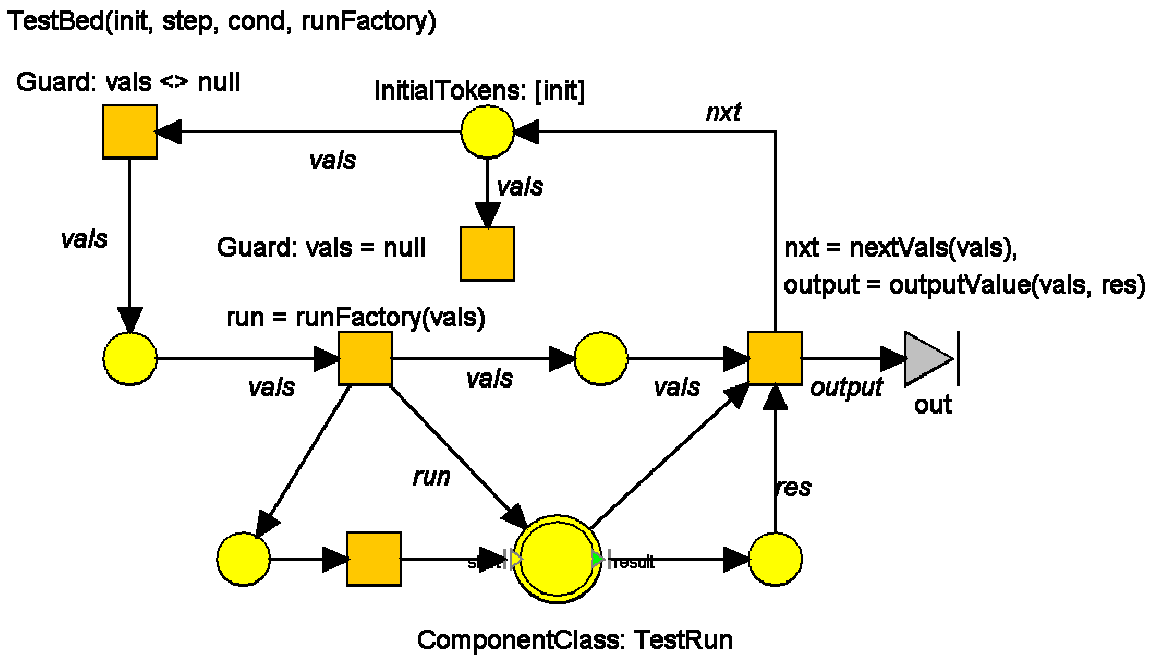

Figure 3. A design-space explorer model (implemented in Moses). The double-lined place accepts and interconnects a model token.

References

1. Overview of the Ptolemy Project, Edward A. Lee, Technical Memorandum UCB/ERL M01/11 March 6, 2001. See also http://ptolemy.eecs.berkeley.edu/.

2. Exploratory Performance Evaluation using Dynamic and Parametric Petri Nets, Robert Esser and Jörn W. Janneck, Proc. High Performance Computing 2000.

3. H. John Reekie, *Realtime Signal Processing: Dataflow, Visual, and Functional Programming*, Ph. D. thesis, School of Electrical Engineering, University of Technology at Sydney, September 1995

4. K. Keutzer, S. Malik, A. R. Newton, J. Rabaey and A.Sangiovanni-Vincentelli. System Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12), December 2000.

5. Thomas A. Henzinger, Benjamin Horowitz, Christoph Meyer Kirsch. Giotto: A Time-Triggered Language for Embedded Programming. Proceedings of EMSOFT 2001. Lecture Notes in Computer Science, Volume 2211, Springer-Verlag, 2001.

6. Luca de Alfaro and Thomas A. Henzinger. Interface theories for component-based design. Proceedings of the First International Workshop on Embedded Software (EMSOFT 2001), Lecture Notes in Computer Science, Springer-Verlag, 2001.

7. E. A. Lee and A. Sangiovanni-Vincentelli, ``A Framework for Comparing Models of Computation,'' IEEE Transactions on CAD, Vol. 17, No. 12, December 1998.